AIO-TFRC: A Light-weight Rate Control Scheme for Streaming over Wireless

Minghua Chen and *Avideh Zakhor*

Department of Electrical Engineering and Computer Sciences
University of California at Berkeley, CA 94720
{minghua, avz}@eecs.berkeley.edu

Abstract— **Rate control is an important issue in video streaming applications for both wired and wireless networks. A widely accepted rate control method in wired networks is TCP Friendly Rate control (TFRC). TFRC assumes that packet loss in wired networks is primarily due to congestion, and as such is not directly applicable to wireless networks in which the main cause of packet loss is at the physical layer. In our previous work, we proposed MULTFRC as an end-to-end based solution to this problem. By opening appropriate number of TFRC connections, MULTFRC not only avoids modifications to the network infrastructure or protocol stack, but also results in full utilization of the wireless channel. In this paper, we propose an alternative end-to-end approach to MULTFRC, denoted by AOI-TFRC. AIO-TFRC enjoys all the advantages of MULTFRC except that it combines multiple connections into one connection in order to reduce implementation complexity, and undesirable consequences of "quantization effect" associated with MULTFRC. NS-2 simulations are carried out to characterize AIO-TFRC's performance, and to show its fairness with respect to TCP. The reduced complexity of AIO-TFRC makes it appealing for actual implementations on mobile handheld devices.**

I. Introduction

Rate control is an important issue for streaming in both wired and wireless networks. On one hand, too low of a streaming rate could underutilize the network bandwidth; on the other hand, too aggressive of a streaming rate could result in serious congestion collapse at the shared bottlenecks [1]. A widely accepted rate control method in wired networks is TCP friendly Rate Control (TFRC) [2]. This is an equation based rate control in which the TCP [1] Friendly rate is computed as a function of packet loss rate, round trip time and packet size. Both TCP and TFRC assume that packet loss in wired networks is primarily due to congestion, and as such are not applicable to wireless networks in which the main cause of packet loss is at the physical layer. In previous work, we have shown that this results in underutilization of the wireless channel [3]. Hence rate control for streaming applications over wireless is still an open problem.

There have been a number of efforts to improve the performance of TCP or TFRC over wireless [4]–[9]. Snoop, a well-known solution, is a TCP-AWARE local retransmission link layer approach [4]. A Snoop module resides on the router or base station on the last hop wireless link, and records a copy of every forwarded packet. Assuming snoop module can access TCP acknowledgement packets (ACK) from the TCP receiver, it looks into the ACK packets and carries out local retransmissions when a packet is corrupted by wireless channel errors. It is easy to modify this scheme to improve TFRC's performance.

End-to-end statistics can also be used to help detect congestion when a packet is lost [5], [6], [8]. The main observations behind these schemes are as follows. First, one way delay between a sender and a receiver increases monotonically if there is congestion. Second, inter-arrival time is expected to increase if there are packet losses caused by wireless channel errors.

Tang et. al. have proposed the idea of using small dummy packets to actively probe the network, so as to differentiate between packet loss due to congestion and that due to channel error [7]. Yang et. al. [9] propose a cross-layer scheme that uses link layer information at the receiver to determine whether a packet loss is caused by wireless channel errors or congestion, assuming only the last link is wireless.

All these approaches either hide end-hosts from discovering packet loss caused by wireless channel error, or provide end-hosts the ability to differentiate between packet loss caused by congestion, and that caused by wireless channel error. They achieve this by either modifying the network infrastructure or the protocol stack, potentially making them hard to deploy in practice.

Our recently proposed approach, MULTFRC [3], improves the performance of TFRC over wireless networks by measuring the round trip time (RTT), and adjusting the number of TFRC connections of the streaming application accordingly. Specifically, it decreases the number of connections multiplicatively if RTT shows an increasing trend, and increases inversely it otherwise. We have shown in [3] that MULTFRC can control the number of connections around the optimal value to achieve the highest throughput and lowest packet loss rate, with no modification to the network infrastructure or the protocol stack. This makes MULTFRC different from all the existing approaches in that it is fairly straightforward to deploy.

Nevertheless, if the optimal number of connections is non-integer, MULTFRC oscillates around the fractional optimal, resulting in both large throughput variability and underutilization. This so called "quantization effect" is most serious when the optimal number of connections is small, e.g. between 1 and

3. Moreover, operating multiple connections in one application requires more resources than one connection; these include ports, memory, feedback signalling, and computation power. These make MULTFRC inefficient, especially for implementation on low power, resource-limited handheld devices.

In this paper, we propose an improved scheme, called ALL-IN-ONE TFRC (AIO-TFRC), to address these drawbacks of MULTFRC. We achieve this by integrating the control law for the number of connections in MULTFRC into one TFRC connection with the same utilization performance as that of MULTFRC. NS-2 simulations are carried out to evaluate AIO-TFRC's performance, and to show its fairness with respect to TCP.

Related work in [10] has proposed improvements to MULTFRC, at the cost of modifying TFRC protocol. It follows the same strategy as MULTFRC in that it controls the way TFRC computes loss event rate, and hence controls the sending rate accordingly. However, this work is still a multiple connection approach, and as such suffers from the quantization effect; furthermore it is more complex than AIO-TFRC, and its fairness with respect to TCP is not evaluated in [10].

The rest of the paper is structured as follows. In Section II, we briefly review MULTFRC. We propose AIO-TFRC in Section III, followed by NS-2 simulation results in Section IV. Section V includes discussions and conclusions.

II. Overview of MULTFRC

We have shown in Theorem 1 in [3] that if the packet loss rate caused by wireless channel error, denoted by p_w, is high enough, then wireless connection suffers from bandwidth underutilization. Furthermore, for a given network setting, there is an optimal number of connections[1] for an application to achieve the highest possible throughput and minimum packet loss rate. Opening more connections than the optimal results in an increase in RTT, and subsequently an increase in end-to-end packet loss rate [3], [11].

A strategy leading to the optimal number of connection, hence the optimal performance, can then be described as: keep increasing the number of connections until an additional connection results in increase of end-to-end RTT or packet loss rate, which indicates the full utilization of the bottleneck links.

Following this strategy, MULTFRC measures the RTT, and adjusts the number of connections so as to (a) utilize the wireless bandwidth efficiently, and (b) ensure fairness between applications. Specifically, it measures the average RTT, denoted by ave_rtt, and Inversely Increases and Additively Decreases (IIAD(α, β)) the number of virtual connections, denoted by n, based on the following law:

$$n = \begin{cases} n - \beta, & \text{if } ave\_rtt - rtt\_min > \gamma\ rtt\_min; \\ n + \alpha/n, & \text{otherwise.} \end{cases} \tag{1}$$

where rtt_min is the minimum ave_rtt seen so far, and α, β, and γ are preset parameters empirically chosen to be $\alpha = \beta = 1, \gamma = 0.25$ [3]. MULTFRC quantizes n to its closest integer number, denoted by $\bar{n}$, and opens multiple TFRC connections accordingly.

[1]Not necessarily an integer.

For a given route, $ave\_rtt - rtt\_min$ corresponds to current queuing delay, and γrtt_min is a threshold on the queuing delay that MULTFRC can tolerate before it starts to decrease n. Thus by evaluating the relation between ave_rtt and rtt_min, MULTFRC detects full utilization of network bottleneck, and controls n accordingly.

In [3], [11], we have evaluated the performance of MULTFRC system through NS-2 simulations and actual experiments over Verizon Wireless 1xRTT CDMA data network. Simulations and experiments have shown that MULTFRC can achieve reasonable utilization of the wireless bandwidth, does not starve applications that use one TCP connection, and significantly improves video streaming performance.

However, there are two drawbacks associated with MULTFRC. First drawback has to do with bandwidth underutilization, and the second one with implementation complexity. We will begin with utilization drawback. NS-2 simulations in [3] show that although MULTFRC performs reasonably well, there is still some gap between its throughput and the optimal. Specifically, MULTFRC achieves only 77% utilization for $p_w = 0.02$. This suboptimal performance has two causes. First one is the control behavior described in (1): as described, n is decreased when the full utilization of bottlenecks is detected, and is inversely increased until the next full utilization is detected. During this period, bottlenecks stay underutilized, resulting in suboptimal average throughput. It is impossible to remove this sub-optimality determined by the control law without changing the law. The second reason for bandwidth underutilization is the "quantization effect" in MULTFRC whereby in practice the number of connections is forced to be an integer. This loss of granularity typically results in bandwidth underutilization. For example, if the optimal number of connections has been determined to be 1.5, then n is forced to take fractional values between 1 and 3, e.g. 1, 1.25, 1.45, 2.14, 1.14, ..., as dictated by (1). MULTFRC then quantizes n to the closest integer to oscillate between one and two, resulting in loss of throughput granularity. This effect can be eliminated by avoiding the quantization step.

The second drawback of MULTFRC is of a more practical nature. Operating multiple connections in one application could potentially consume too much system resources. For example, each TFRC connection uses a different port to send out data packets, carries out individual feedback process, and updates the loss event rate and RTT even though they are highly correlated for these TFRC connections. Clearly, there is unnecessary overhead associated with operating multiple connections, in terms of computation, processing power, memory, and ports, particularly for today's low power, resource-limited handheld devices.

III. ALL-IN-ONE TFRC (AIO-TFRC

In this section, we propose an alternative to MULTFRC, called ALL-IN-ONE TFRC (AIO-TFRC), in order to address the two drawbacks of MULTFRC, while retaining the same control law for n as in MULTFRC.

To achieve this goal, we integrate the bandwidth filtered loss detection (BFLD) technique from [12], to be described shortly, together with the control law in (1) to construct the AIO-TFRC system. The system framework is shown in Fig. 1. Basically, the Sink at the receiver feeds back the RTT and loss event rate to the sender. The sender then adjusts of n based on (1), and sends out the data packets at a rate of n times that of one TFRC's sending rate. The functionalities of AIO-TFRC senders and receivers are described as follows:

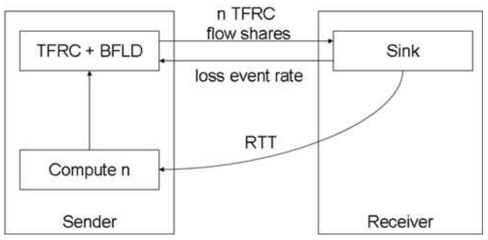

Fig. 1. The system framework of AOI-TFRC.

- *Sender*: There are two functional components in the sender. One component is represented by the "compute n" block. It receives the RTTs from the receiver, computes an ave_rtt, by averaging these RTT samples over a 20 second window, then updates n according to (1) every 20 seconds, i.e. using the same law as MULTFRC. For AIO-TFRC, we choose $\alpha = \beta = 1$, and $\gamma = 0.5$.
  The other component is represented by the "TFRC+BFLD" block, and it has two functionalities: first, it obtains the updated n from the "compute n" component, as well as the loss event rate from the receiver. It then computes the TCP friendly rate of one TFRC connection as the standard TFRC does [2], and adjusts the sending rate to be n times that of one TFRC. The second functionality of the "TFRC+BFLD" block in the sender is to mark the headers of selected data packets before they are sent out. The data packets to mark are selected in such a way that they form a virtual single TFRC flow, and hence correspond to $1/n$ of all the outgoing packets. For example, if $n = 1.5$, then "TFRC+BFLD" evenly marks $2/3$ of all outgoing packets. The reason for the marking is to facilitate the loss event rate measurement at the receiver.
- *Receiver*: The AIO-TFRC Sink component reports the RTT and the loss event rate of the virtual TFRC connection to the sender every RTT. The only difference between the AIO-TFRC Sink and the original TFRC Sink is that the AIO-TFRC Sink measures and updates the loss event rate based on the virtual TFRC flow with marked packets.

We now explain the reasoning behind the marking process. It has been argued in [12] that if the sending rate of the application is adjusted to be n times that of one TFRC, then TFRC Sink at the receiver underestimates the loss event rate based on using all the received packets. TFRC Sink records the beginning of a loss event when a packet loss is detected. The loss event ends when, after a period of one RTT, another packet loss is detected. A loss event interval is defined as the difference in sequence numbers between the above two lost packets; the loss event rate is thus estimated by taking the inverse of this difference. In the case where all the packets are used to estimate loss event rate, since the number of packets received by TFRC Sink is n times that of one TFRC, the above procedure tends to underestimate the loss event rate. To overcome this problem, we sample the outgoing packets at the sender to form a single virtual TFRC stream at the sender, and modify TFRC Sink to carry out the loss event rate measurement based on this virtual stream. This is the functionality of BFLD as verified in [12].

We now explain the reason to choose a larger γ in AIO-TFRC than in MULTFRC. MULTFRC achieves n TFRC flow shares by opening $\bar{n}$ independent TFRC connections, while AIO-TFRC achieves n flow shares by opening one connection and sending at n times the sending rate of one TFRC connection. In situations where the throughput of each TFRC connection is a function of the random packet loss rate, e.g. that caused by random wireless channel error, it is well known the latter method results in a higher variance in the aggregate sending rate. Since the queueing delay along the route is a function of the aggregate sending rate, AIO-TFRC experiences higher queuing delay variance along the path. Therefore, in order to achieve the same level of confidence in the measured queueing delay, used to adjust n in (1), AIO-TFRC needs a larger threshold than MULTFRC. In this paper, we have empirically chosen $\gamma = 0.5$.

IV. Simulation results

In this section, we carry out NS-2 simulations to evaluate the performance of AIO-TFRC. Specifically, we examine three issues in these simulations: (a) how AIO-TFRC performs in terms of average throughput, average RTT, and packet loss rate, as a function of p_w, and how it compares with MULTFRC; (b) whether n is stable; (c) whether or not AIO-TFRC is fair to TCP. In all the simulations, throughput is measured every 10 seconds, packet loss rate is measured every 30 seconds, the average RTT is measured every 100 packets, and the number of connections is sampled whenever there is a change.

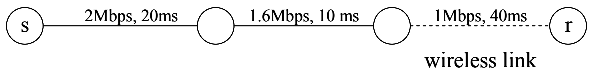

Fig. 2. The simulation topology for AIO-TFRC's utilization evaluation.

The topology used in simulations for utilization evaluation is shown in Fig. 2. The sender is denoted by s, and the receiver is denoted by r. They both run AIO-TFRC at the application layer. For the simulations, the wireless bandwidth, denoted by B_w, is set be 1 Mbps and is assumed to be the bottleneck. The wireless link is modeled by an exponential error model, and the wireless packet loss rate p_w varies from 0.0 to 0.08

in increments of 0.02. DropTail type queue is used for each node.

We simulate the AIO-TFRC system to stream for 9000 seconds. The average throughput, end-to-end packet loss rate, average RTT, and average n for p_w =0.0, 0.02, 0.04, 0.06 and 0.08 are shown in Fig. 3, where $RTT_{min} = 168\ ms$. As seen, the throughput is within 15% of the optimal, and the packet loss rate is almost identical to the optimal, i.e. a line of slope one as a function of wireless channel error rate. As expected, the average n increases with wireless channel error rate, p_w. [2]

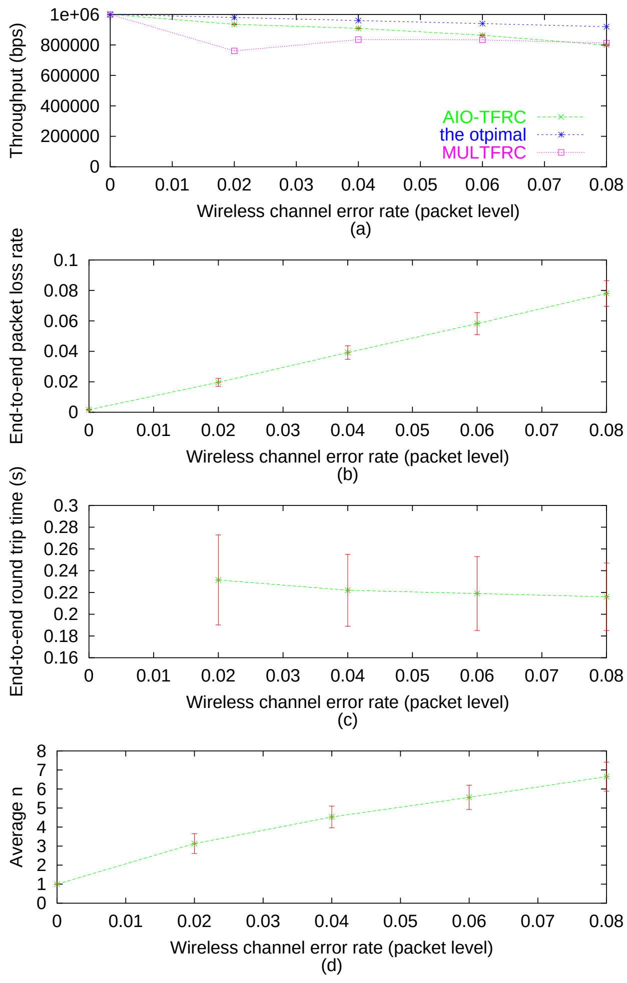

Fig. 3. NS-2 simulations for Bw = 1 Mbps and RTTmin = 168 ms; (a) throughput, (b) end-to-end packet loss rate, (c) end-to-end RTT, (d) number of connections, all as a function of packet error rate on the wireless channel.

The throughput of MULTFRC is also shown in Fig. 3(a) for comparison. As seen, AIO-TFRC has almost the same throughput as MULTFRC when p_w is high, while it significantly outperforms MULTFRC when p_w is low. For example, when $p_w = 0.02$, AIO-TFRC achieves 95% utilization of the wireless bandwidth, while MULTFRC's utilization is only 77%. Therefore, by avoiding the "quantization effect", AIO-TFRC achieves better throughput performance than MULTFRC.

[2]Note the RTT for $p_w = 0$ is not shown in Fig. 3 because it represents the channel error free case in which MULTFRC reduces to one TFRC connection.

To examine the dynamics of AIO-TFRC systems, we show throughput, the number of virtual connections n, packet loss rate, and average RTT as a function of time for $p_w = 0.04$ in Figure 4. As seen, the throughput and the number of connections are quite stable; as expected, packet loss rate is around 0.04 and RTT is properly controlled to be around γRTT_min. Similar results are obtained for other values of p_w.

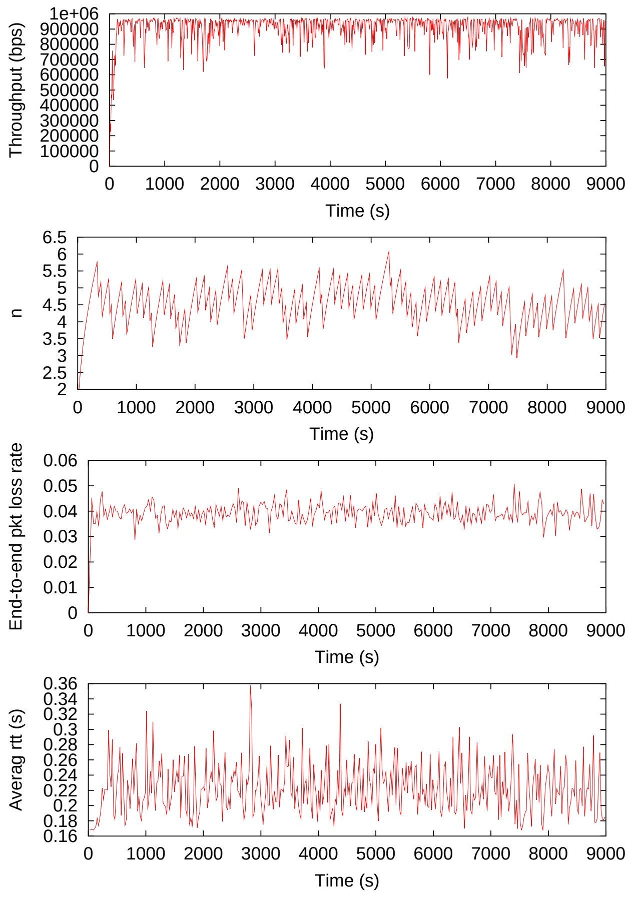

Fig. 4. NS-2 simulations for $B_w = 1Mbps$ and $p_w = 0.04$; (a)throughput, (b) number of connections , (c) end-to-end packet loss rate, (d) end-to-end RTT, all as a function of time.

To investigate the fairness of AIO-TFRC, we carry out NS-2 simulations based on the "dumbbell" topology shown in Fig. 5. Senders are denoted by $si, i = 1, \ldots, 16$, and receivers are denoted by $di, i = 1 \ldots, 16$. We investigate two types of fairness: the inter-protocol fairness between AIO-TFRC and TCP, and the intra-protocol fairness within AIO-TFRC.

The intra-protocol fairness is defined as the fairness between AIO-TFRC flows. In our simulations, we run AIO-TFRC on all 16 sender-receiver pairs shown in Fig. 5 for 5000 seconds, and compare their throughput. AIO-TFRC is said to be intra-

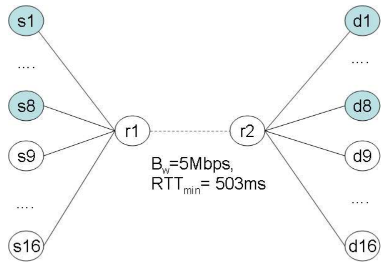

Fig. 5. The simulation topology for AIO-TFRC's fairness evaluation.

protocol fair if all receivers get the same throughput. The fairness ratios for $p_w = 0.01$ and $p_w = 0.04$ are shown in Table I. The fairness ratio is defined as receivers' throughput divided by the average throughput; the closer to one, the more fair the AIO-TFRC system is. As seen, the fairness ratio is fairly close to one, indicating AIO-TFRC flows are fair to each other, at least in this simulation setting. The bandwidth utilization ratios are 95% for $p_w = 0.01$ and 98% for $p_w = 0.04$.

TABLE I
SIMULATION RESULTS FOR INTRA-PROTOCOL FAIRNESS OF AIO-TFRC.

receiver	fairness ratio p_w=0.01	fairness ratio p_w=0.04	receiver	fairness ratio p_w=0.01	fairness ratio p_w=0.04
d1	1.00	0.99	d9	1.02	1.03
d2	0.99	0.99	d10	0.98	0.98
d3	0.96	1.00	d11	0.97	1.01
d4	0.99	0.97	d12	0.99	0.99
d5	1.05	0.95	d13	0.99	1.01
d6	1.04	1.01	d14	1.01	0.97
d7	1.03	1.02	d15	1.01	0.98
d8	1.02	1.03	d16	0.96	1.02

The inter-protocol fairness is defined as the fairness between AIO-TFRC and TCP[3]. In our simulations, we run AIO-TFRC on the first 8 sender-receiver pairs, i.e. $(si, di), i = 1, \ldots, 8$, and TCP on the remaining 8 sender-receiver pairs shown in Fig. 5; each session lasts 5000 seconds, and we compare their throughput for $p_w = 0.01$ and $p_w = 0.02$. Under the simulation settings, each AIO-TFRC consumes more bandwidth than one TCP under full utilization. This is because in this case, the wireless channel error rate is large enough to make the number of virtual connections of each AIO-TFRC to be larger than one. Hence, it is meaningless to define the fairness between AIO-TFRC and TCP as having the same throughput.[4] As such, in our simulations, we define AIO-TFRC to be fair to TCP if it does not result in a decrease in TCP's throughput. Specifically for our simulations, it implies TCP retains the same throughput whether or not it coexists with AIO-TFRC under the same network setting. The throughput of AIO-TFRC and TCP, as well as the total bandwidth utilization ratios for the setup shown in Fig. 5, are shown in Table II for two scenarios: (a) 8 AIO-TFRC coexisting with 8 TCP connections, (b) 16 TCP connections. Figs. 6 and 7 also show the dynamics of throughput, packet loss rate, RTT, and the number of virtual connections n. Comparing AIO-TFRC+TCP with TCP-alone, we see the former has a much higher utilization of the wireless bandwidth at the expense of lower TCP throughput. A careful examination of Fig. 6 reveals that this throughput drop is caused by the higher RTT for AIO-TFRC+TCP as compared with TCP-alone. For example, for $p_w = 0.01$ and $\gamma = 0.5$, AIO-TFRC+TCP experiences around 0.6 seconds RTT, while TCP-alone only experiences 0.5 seconds RTT, i.e. the propagation delay. As TCP's throughput is known to be inversely proportional to RTT, the 20% increase in the RTT explains the 16% decrease in the TCP's throughput shown in first row in Table II.

[3]We use TCP SACK implementation in simulations.

[4]Obviously, there are situations in which AIO-TFRC ends up with performing similar to one TFRC. An example would be AIO-TFRC competing for bandwidth with TCP on wired networks. In that case, however, the fairness between AIO-TFRC and TCP is reduced to the fairness between TFRC and TCP, and has been well explored in [2].

TABLE II
SIMULATION RESULTS FOR FAIRNESS BETWEEN AIO-TFRC AND TCP.

settings	8 AIO-TFRC + 8 TCP			16 TCP	
	ave. thput. (AIO-TFRC) (kbps)	ave. thput. (TCP) (kbps)	utilization (%)	ave. thput. (TCP) (kbps)	utilization (%)
p_w=0.01 γ=0.5	436.501	168.048	98	200.168	65
p_w=0.01 γ=0.1	379.656	185.286	91	200.168	65
p_w=0.02 γ=0.5	486.821	120.313	99	139.674	46
p_w=0.02 γ=0.1	449.226	130.953	95	139.674	46

This increase in the RTT is, by design, a consequence of AIO-TFRC controlling the number of virtual connections n according to (1). As n is only decreased after the queuing delay exceeds the threshold γrtt_min, round trip time is increased when AIO-TFRC increases n to achieve full utilization. One way to address this problem is to use a smaller value for γ, in order to reduce the increase in the RTT, and hence minimize the TCP's throughput drop. However, smaller values of γ also results in lower bandwidth utilization due to increased sensitivity of AIO-TFRC to RTT measurements. As shown in Table II, $\gamma = 0.1$ results in a smaller drop in the TCP's throughput than $\gamma = 0.5$.

V. DISCUSSION AND CONCLUSION

In this paper, we have proposed AIO-TFRC to enhance our previously proposed MULTFRC scheme to address two drawbacks. The first one is the "quantization effect", the second one is the control overhead of operating multiple connections. AIO-TFRC achieves these goals by creating one connection whose throughput is equivalent to that of the optimal number of TFRC connections even though the optimal number could be non-integer. It does so by measuring round trip times, adjusting the number of virtual connections based on the measurements, and using BFLD technique within one connection. NS-2 simulations show that AIO-TFRC achieves

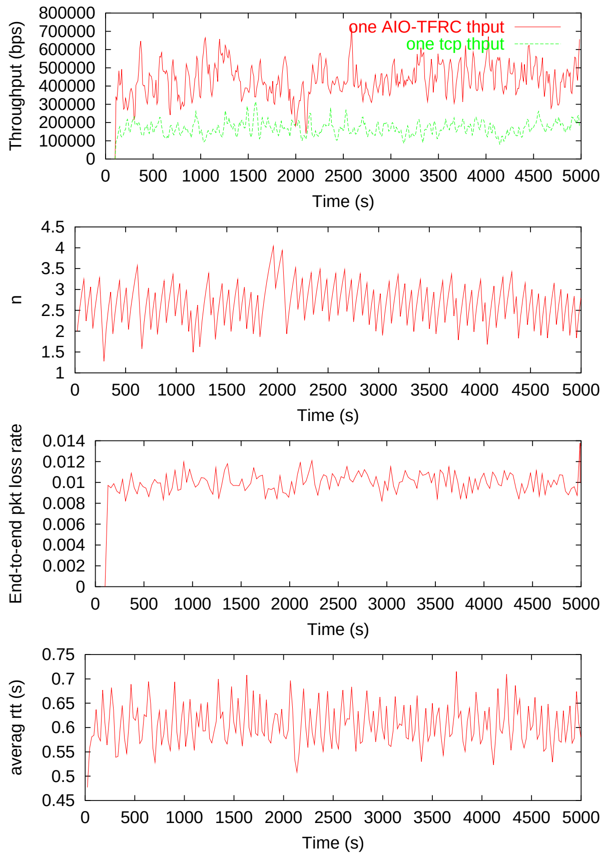

Fig. 6. NS-2 simulation results for the case $p_w = 0.01, \gamma = 0.5$: the dynamics of (a)throughput, (b) number of connections , (c) end-to-end packet loss rate, (d) end-to-end RTT, all as a function of time.

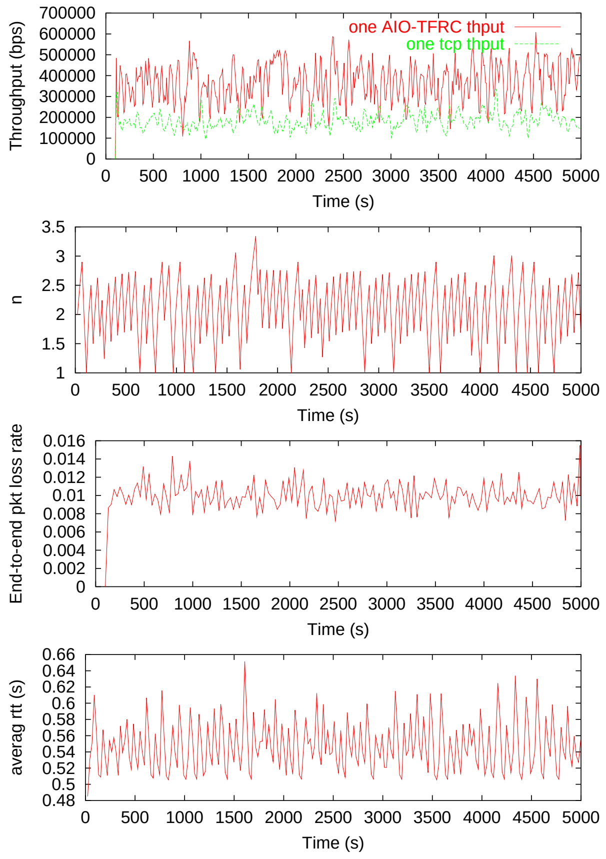

Fig. 7. NS-2 simulation results for the case $p_w = 0.01, \gamma = 0.1$: (a)throughput, (b) number of connections , (c) end-to-end packet loss rate, (d) end-to-end RTT, all as a function of time.

similar throughput as MULTFRC in high packet loss rate situation, but better throughput than MULTFRC at the low packet loss rate scenarios. The simulations also shows that AIO-TFRC is relatively fair to TCP, and highly fair to itself.

Future work includes reducing the throughput variance of AIO-TFRC. This can be done by first measuring multiple instances of loss event rates over multiple virtual streams separately, and then computing aggregate sending rate as the sum of the individual sending rates. The cost associated with this approach is the computational power and memory of handheld devices. It would also be interesting to explore the effect of γ on the trade-off between AIO-TFRC's fairness with respect to TCP and the utilization of wireless bandwidth.

Acknowledgement

The authors would like to thank Ketan Mayer-Patel for mentioning BFLD technique to us at the INFOCOM 2004 conference.

References

[1] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM*, Stanford, CA, Aug. 1998, pp. 314–329.

[2] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. ACM SIGCOMM*, Stockholm, Sweden, Aug. 2000, pp. 43–56.

[3] M. Chen and A. Zakhor, "Rate control for streaming video over wireless," in *Proc. IEEE INFOCOM*, Hongkong, China, Mar. 2004.

[4] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz, "A comparison of mechanisms for improving tcp performance over wireless links," *IEEE/ACM Trans. Networking*, vol. 5, no. 6, pp. 756–769, 1997.

[5] N. Samaraweera, "Non-congestion packet loss detection for tcp error recovery using wireless links," *IEE Proceedings of Communications*.

[6] S. Cen, P. Cosman, and G. Voelker, "End-to-end differentiation of congestion and wireless losses," *IEEE/ACM Trans. Networking*, vol. 11, no. 5, pp. 703–717, 2003.

[7] J. Tang, G. Morabito, I. F. Akyildiz, and M. Johnson, "Rcs: A rate control scheme for real-time traffic in networks with high bandwidth-delay products an high bit error rates," in *Proc. IEEE INFOCOM*, Alaska, USA, Apr. 2001, pp. 114–122.

[8] G. Yang, M. Gerla, and M. Y. Sanadidi, "Adaptive video streaming in presence of wireless errors," in *Proc. ACM MMNS*, San Diego, USA, Jan. 2004.

[9] F. Yang, Q. Zhang, W. Zhu, and Y.-Q. Zhang, "End-to-end tcp-friendly streaming protocol and bit allocation for scalable video over mobile wireless internet," in *Proc. IEEE INFOCOM*, Hongkong, China, Mar. 2004.

[10] X. Tong and Q. Huang, "Multfrc-lerd: An improved rate control scheme for video streaming over wireless," in *Proc. 5th Pacific Rim Conference on Multimedia*, Tokyo, Japan, Nov. 2004, pp. 282–289.

[11] M. Chen and A. Zakhor, "Transmission protocols for streaming video over wireless," in *Proc. ICIP*, Singapore, Oct. 2004, pp. 1743–1746.

[12] D. E. Ott, T. Sparks, and K. Mayer-Patel, "Aggregate congestion control for distributed multimedia applications," in *Proc. IEEE INFOCOM*, Hongkong, China, Mar. 2004.